Functional EIAO DW, R2.2

Deliverable 6.3.3.1-3

Thomsen, Christian; Pedersen, Torben Bach; Frøkjær, Jens; Mayeli, Massih

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

EIAO
European Internet Accessibility Observatory
Project no.: IST-2003-004526- STREP

Functional EIAO DW, R2.2

Deliverable Number: 6.3.3.1-3

Version:	1.0.1
Date:	2008-05-13
Author:	Christian Thomsen, Torben Bach Pedersen, Jens Frøkjær, Massih Mayeli
Dissemination Level:	
Status:	FINAL

This document consists of 25 pages plus this cover

Version Control

Version	*Status*	*Date*	*Change*	*Author*
0.1	RC	2008-03-04	Revising the document	Christian Thomsen, Massih Mayeli
1.0	FINAL	2008-04-30	Corrections from proof reading	Christian Thomsen
1.0.1	FINAL	2008-05-13	Minor updates based on comments from Nils Ulltveit-Moe	Christian Thomsen

Table of Contents

1 Introduction....3
1.1 Brief project description....3
1.2 Scope of this document....3
1.3 Related work and readers instructions....4
2 Conceptual model for EIAO DW....5
3 Logical Model....7
4 Physical model....10
4.1 Naming Conventions....10
4.2 Physical Schema....10
5 Loading of the data warehouse....13
5.1 The ETL process....15
5.1.1 Schema creation and prefilling of tables....15
5.1.2 Loading the datastaging schema....16
5.1.3 Filling the eiaodw schema....18
6 C-WAM implementation in stored procedures....19
6.1 The implemented procedures....19
6.2 Materialized views for the implemented procedures....26
7 A Note on The EIAO DW Specification....27
7.1 The Challenge of EIAO DW Specification....27
7.2 Modularity....27
7.3 Extensibility....27
7.4 Scalability....28
7.5 Maintainability....28

1 Introduction

1.1 Brief project description

The overall objective of the project is to contribute to better e-accessibility for all citizens and to increase use of standards for on line resources. The project has been carried out as part of the Web Accessibility Benchmarking cluster (WAB) together with the projects SupportEAM and BenToWeb.

The project establishes the technical basis for a possible European Internet Accessibility Observatory (EIAO) consisting of:

- A set of web accessibility metrics.
- An Internet robot for automatic and frequent collecting of data on web accessibility and deviations from web standards (the WAI guidelines)
- A data warehouse providing on line access to collected accessibility data.

1.2 Scope of this document

This document covers the schema design for release 2.2 of the data warehouse in the EIAO project, the EIAO DW. The schema design follows the classic approach with conceptual, logical, and physical models as described, e.g, in R. Elmasri and S. Navathe: "Fundamentals of Database Systems", Addison Wesley, 2003.

The first part of this document (Chapter 2 and 3) includes illustration of the conceptual model and description of the logical model. The conceptual model is describing the entities and their relationships, but without using a specific data model. In the logical model the relational data model is used and, thus, the logical model shows how the user will interact with the implementation of the database. That means that Part 1 of this document is what an "**external user**" will need to read to use the EIAO DW. A detailed description of the conceptual model can be found in document D6.1.1.1-3 Appendix A.

The physical model that shows how the data is actually stored on the disk is described in the second part of this document. This includes descriptions of indexes and internal structures. That part of the document also includes a description of another database schema only used by the ETL tool as well as a description of how the ETL process works. So the second part of the document (Chapter 4) serves as documentation for the more technical parts and should only be used by the **data warehouse developers**.

The last part of the document describes the stored procedures that implement the C-WAMs described in D3.3.2. Thus the third part (Chapter 6) of the document is of interest to the **user interface and report developers**.

1.3 Related work and readers instructions

This document is related to the following documents:

- D3.3.2 "Final version of EIAO WAMs" describes the Web Accessibility Metrics (WAMs) that generate the data to store in the data warehouse. Further the C-WAMs that are implemented as stored procedures are described in D3.3.2.

- D6.6.1.2.2 “Collecting and maintaining a URL directory for the project” describes the URL directory used in the project.
- The RDF schema describing the data present in the RDF databases has a major impact on the ETL. The RDF schema is available in the SVN repository from http://svn.eiao.net/robacc/RDFSchema/RDFSchema.rdf
- The schema for the URL repository which is implemented in PostgreSQL also affects the ETL. Description for this schema can be found at http://svn.eiao.net/robacc/URL_repository/tables.sql
- Functional specification and architecture of ROBACC DW are described in D6.1.1.1-2 Appendix A

2 Conceptual model for EIAO DW

In this chapter we illustrate the conceptual model for EIAO DW. In the following chapters the logical and physical models will be described.

The conceptual model is shown in Figure 2.1. The notation is based on UML 2.0 (see www.uml.org). The model illustrates classes for which information should be stored in the EIAO DW. In the model, attributes of the classes are shown as well as associations between different kinds of classes.

The dotted ellipses are strictly speaking not part of the conceptual model, but are included for clarity. They show how the classes are grouped together as *dimensions* in the logical model (see Chapter 3). In the ellipses, the hierarchy within each dimension is represented such that higher levels in the hierarchy are drawn above lower levels in the hierarchy. For example, in the Date dimension it is seen that days roll up into months.

For detailed explanation of each of the shown classes, its attributes, and its associations to other classes please refer to document D6.1.1.1-3.

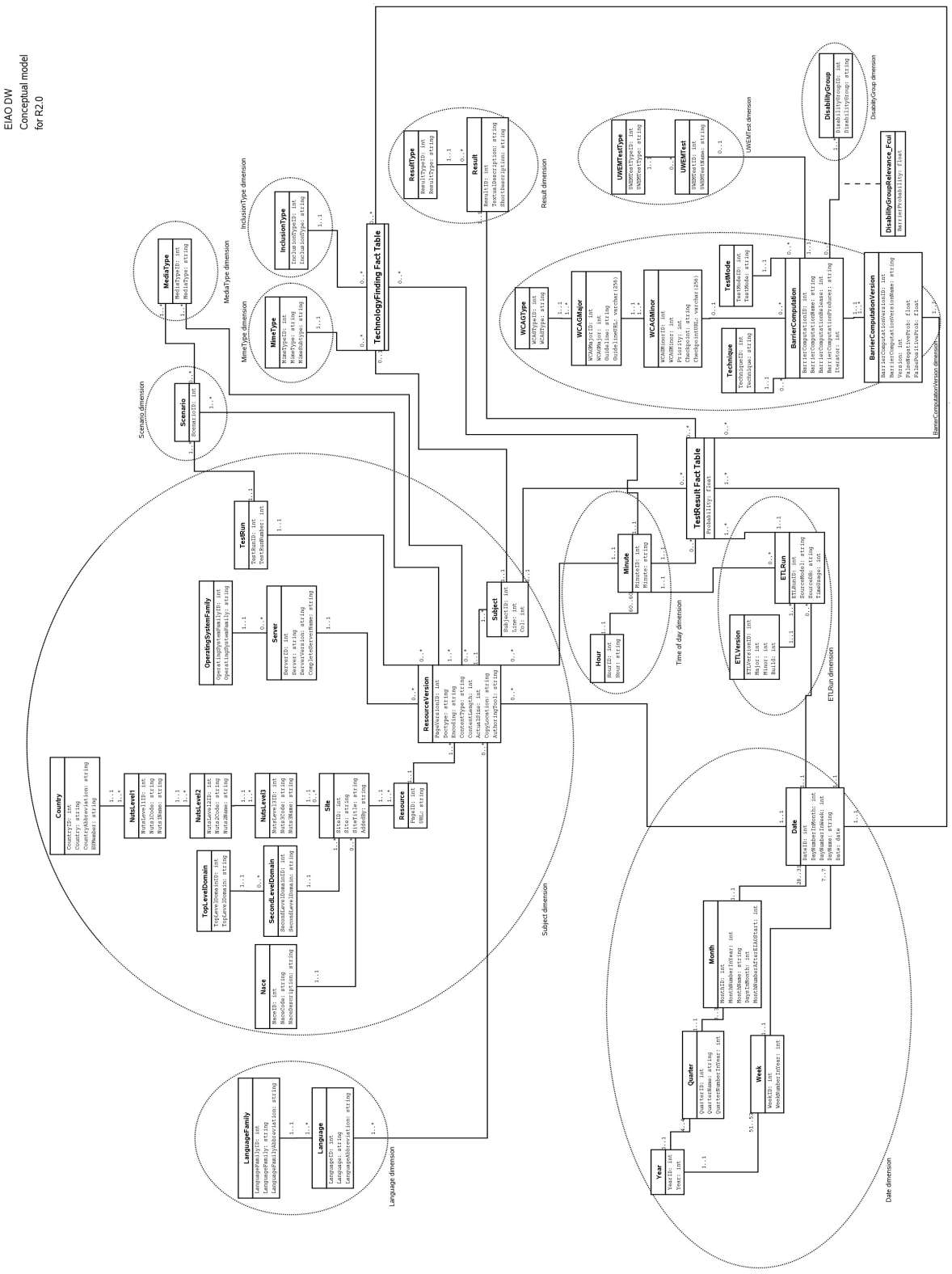

Figure 2.1Conceptual model

3 Logical Model

In this chapter we present the logical model for the data warehouse. The logical model is shown in Figure 3.1. This is the schema that the user sees when interacting with the data warehouse. Thus this model is a *relational* model and data types are shown as SQL data types.

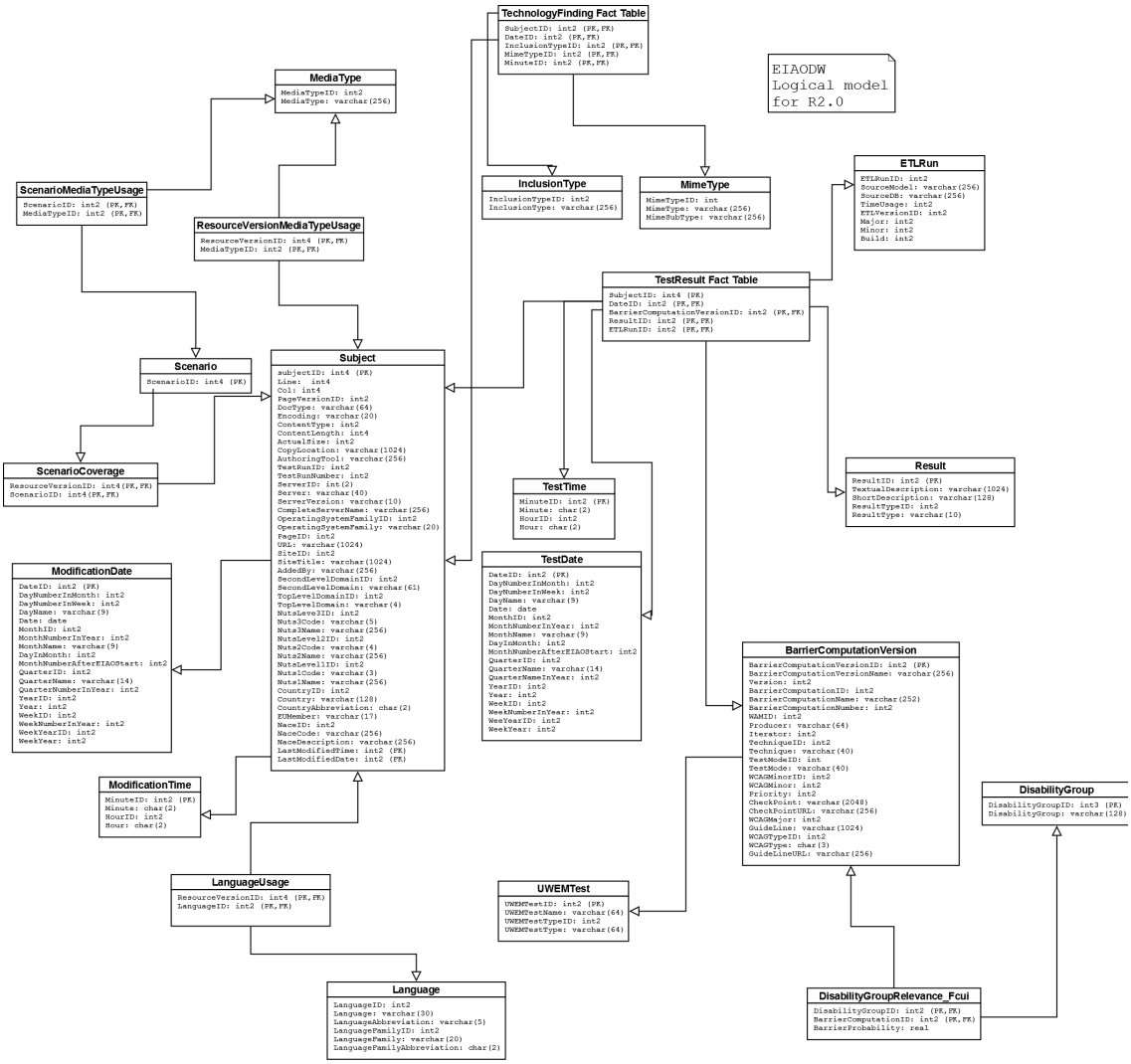

Figure 3.1 Logical model

Basically, the logical model is derived by forming a dimension for each dotted ellipse in the conceptual model. The dimension then has attributes corresponding to the attributes of the classes within the ellipse. Further, some so-called bridge tables for representing many-many relationships have been added. These are ResourceVersionMediaTypeUsage, ScenarioMediaTypeUsage, ScenarioCoverage, and LanguageUsage. ScenarioCoverage is used to represent which scenarios a subject belongs to. It should be noticed that the dimension tables described here are implemented as views. For each of these views, there is currently a base table in the physical model which holds exactly the same data. The reason for the use of views is that it provides flexibility for changes later on. The user of the DW should not be concerned about this and can just pretend that all dimensions in the logical model exist as tables in the physical model. For that reason we will continue to use the term “table” in this chapter, even though we actually are talking about views.

As for the conceptual model, the ID attributes do not carry any meanings, but are merely used as a mean to identify different dimension values. For each dimension table, the ID attribute from the class lowest in the hierarchy in the conceptual model is used as the primary key for the dimension table (each primary key is marked by “(PK)”). The TestResult fact table has a foreign key to each of the Subject, Result, TestDate, TestTime, and BarrierComputationVersion dimension tables (foreign keys are marked by “(FK)” and on Figure 3.1 shown as arrows towards the referenced attributes). Together these foreign keys constitute the primary key of the TestResult fact table. The foreign key MinuteID to the TestTime dimension represents the time the given result was obtained. Similarly, the foreign key DateID to the TestDate dimension, represents the date a result was obtained. Finally the TestResult table has foreign keys, SubjectID, ResultID, and BarrierComputationVersionID, to the Subject, Result, and BarrierComputationVersion dimensions. These foreign keys represent what subject was tested, the outcome of the test, and the barrier computation version performing the test, respectively.

The Subject dimension table has foreign keys referencing the two outriggers ModificationDate and ModificationTime to represent the last modification date and time of the page of the subject, respectively. Notice that TestTime and ModificationTime are similar. In the implementation, both of them are defined as views selecting the same data from one underlying table. The same holds for TestDate and ModificationDate. The reason for the use of two tables to represent dates and two tables to represent times is that it makes it easier to formulate queries where the user puts constraints on both when the page was modified and when the page was tested.

The Subject dimension also has two other outriggers. These are Scenario and Language that represent which scenarios and language a subject belongs to, respectively. Notice that the relationships are modeled by means of bridge tables referencing the outriggers and the Subject dimension.

Also the BarrierComputationVersion dimension has an outrigger: UWEMTest.

The MediaType dimension has two outriggers: Scenario and ResourceVersion. These are respectively modeled by means of bridge tables ScenarioMediaTypeUsage and ResourceVersionMediaTypeUsage.

Apart from what is described above, attributes have the same names and purposes in the logical model as in the conceptual model (described above). The SQL data types have been

included in Figure 3.1. The used types are 2 and 4 bytes integers (int2 and int4, respectively) and varchars. In a few places where a string always has a fixed length, char is used to indicate this (note that opposed to many other DBMSs the use of char instead of varchar does not give any performance gain in PostgreSQL).

For an attribute marked with “(FK)”, for foreign key, a row can only hold a specific value for this attribute if there is a row with this value for the referenced attributes pointed towards by the arrow. It is, though, not required here that only primary or unique keys are referenced by foreign keys, In particular, the SiteID of the Subject-dimension is referenced even though many rows can hold the same values for SiteID. Since PostgreSQL does not support this, this relationship will not be declared as a foreign key in the implementation, but it can be used as if it were.

4 Physical model

In this chapter the physical model for EIAO DW R2.2 is presented.

4.1 Naming Conventions

In this section, we describe the naming conventions used in the physical model. All tables will be given names that end in "_T" (i.e., an underscore followed by the letter T). Views, on the other hand, do not have a common suffix. The reason for this is that views are to be used externally, i.e., the views are what the user sees. Therefore these names should be intuitive and simple.

An index will be given the name of the table it is defined on followed by an underscore which again is followed by the attributes (separated by underscores) the index is defined on. Finally, the index ends with "_IDX". Thus, the pattern for an index name is *Tablename_Attribute1_Attribute2_..._AttributeN*_IDX. These names can thus be very long, but it is easy to see what an index is defined on. When the names get so long that PostgreSQL truncates them, abbreviations are used for the attribute names. The table names will not be abbreviated. Notice that the indexes created implicitly by PostgreSQL for primary keys are named *Tablename*_pkey.

4.2 Physical Schema

The schema for the physical model is identical to the schema for the logical model apart from the following differences. Only one table, TimeOfDay_T, to hold times will be defined. Also, only one table, Date_T, to hold dates will be defined. The reason for this is, as previously mentioned, that additional views (TestTime and ModificationTime on TimeOfDay_T and TestDate and ModificationDate on Date_T) will be created. Thus, the physical schema looks as shown in Figure 4.1.

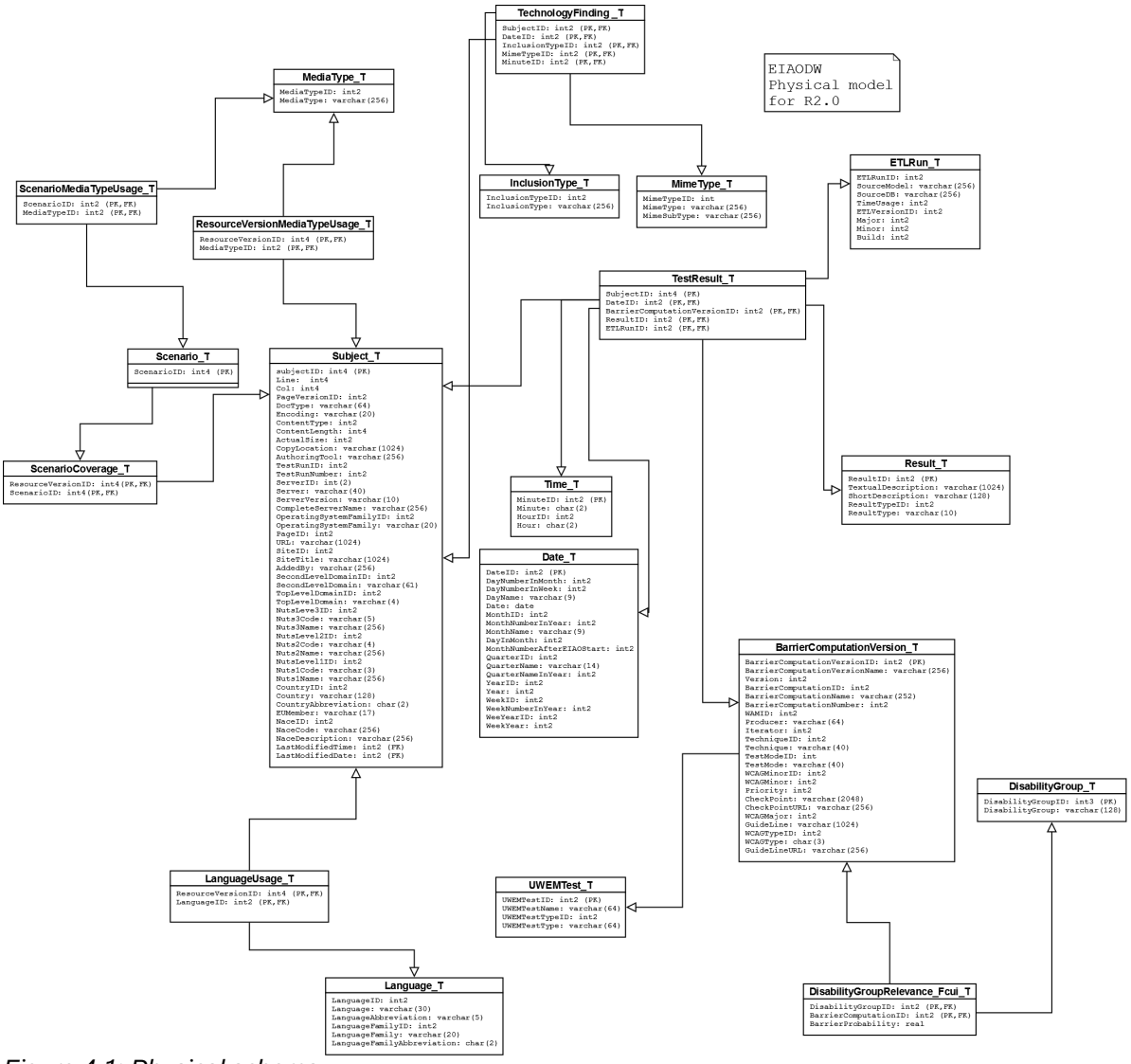

Figure 4.1: Physical schema

For each table *X* in the logical model (except TestDate, TestTime, ModificationDate, and ModificationTime already described), a table named *X*_T exists in the physical model. A view named *X* is then also defined. The reason for this solution is that it provides flexibility for redefinitions later on.

For efficiency reasons the indexes on the following attributes or pairs of attributes are also included in the physical model.

- TestResult_T_SubjectID_IDX on SubjectID in TestResult_T
- TestResult_T_BarrierComputationVersionID_IDX on BarrierComputationVersionID in TestResult_T
- TestResult_T_ResultID_IDX on ResultID in TestResult_T

- Date_T_MonthNumberInYear_DateID_IDX on (MonthNumberInYear, DateID) in Date_T
- Date_T_MonthName_DateID_IDX on (MonthName, DateID) in Date_T
- Date_T_QuarterName_DateID_IDX on (QuarterName, DateID) in Date_T
- Date_T_QuaterNumberInYear_DateID_IDX on (QuarterNumberInYear, DateID) in Date_T
- DateT_Year_DateID_IDX on (Year, DateID) in Date_T

- Subject_T_Site_IDX on (Domain, SubjectID) in Subject_T
  Subject_T_ResourceVersion_IDX on (TestrunID, ResourceVersionID, SubjectID) in Subject_T
- Subject_T_CountryAbbreviation_SubjectID_IDX on (CountryAbbreviation, SubjectID) in Subject_T
- Subject_T_NutsLevel1_SubjectID_IDX on (TestRunID, Nuts1Code , SubjectID) in Subject_T
- Subject_T_NutsLevel2_SubjectID_IDX on (TestRunID, Nuts2Code , SubjectID) in Subject_T
- Subject_T_NutsLevel3_SubjectID_IDX on (TestRunID, Nuts3Code , SubjectID) in Subject_T

- BarrierComputationVersion_T_BarrierCompID_BarrierCompVerID_IDX on (BarrierComputationID, BarrierComputationVersionID) in BarrierComputationVersion_T
- BarrierComputationVersion_T_WCAGMaj_WCAGMin_BarrierCompVerID_IDX on (WCAGMajor, WCAGMinor, BarrierComputationVersionID) in BarrierComputationVersion_T

Notice that in the pairs of attributes, the primary keys for the dimension tables are always included. By using these covering indexes, it it is fast to locate the relevant facts in the fact table.

Further, an implicit index exists for each primary key (these implicit indexes are automatically created by PostgreSQL).

This schema is relatively easy to extend. This is done by partitioning the sub-tables inheriting from ScenarioCoverage_T, ScenarioMediaTypeUsage_T, Scenario_T, Subject_T, TestResult_T and TechnologyFinding_T tables that hold the data for a month. Also more materialized views and bitmap indexes can be added later. The tables in the physical model are located in the schema[1] *eiaodw* in the database. In the database, the schema *datastaging* also exists. The *datastaging* schema is used by the ETL application during load of the DW. Finally the schema *matviews* exists. This schema holds a number of materialized views used by the stored procedures calculating the CWAMs.

5 Loading of the data warehouse

As previously mentioned, the *datastaging* schema is used when the DW is loaded. This schema has a table for each class in the conceptual model previously described. In the tables, the ID attributes are used as primary keys. Note, however, that to improve the load performance, constraints (both primary key and foreign key constraints) are not declared

1 Notice that the term “schema” here refers to a schema in the PostgreSQL database, i.e. a subpart of the database so to speak.

when the *datastaging* schema is being loaded. The constraints are not declared until *after* all the tables have been filled with data. The reason for this is that it is more efficient to load the data without considering the constraints. When the data is loaded, the constraints are added (this is then relatively fast) to ensure that the data is consistent.

Each of the tables has attributes as its corresponding class in the conceptual model. Further, associations in the conceptual model are represented by foreign keys from one of the participating tables to the other. The foreign keys are always named as the primary keys they reference. Notice again that the constraints are not declared until after the load.

Graphically the *datastaging* schema looks as shown in Figure 5.1.
Further, the following indexes are declared in the *datastaging* schema. The naming conventions are as previously described for the logical model.

- TopLevelDomain_TopLevelDomain_IDX on TopLevelDomain in TopLevelDomain
- SecondLevelDomain_SecondLevelDomain_IDX on SecondLevelDomain in SecondLevelDomain
- Site_DomainID_IDX on SecondLevelDomainID in Site
- Resource_URL_IDX on URL in Resource
- LanguageFamily_LanguageFamilyAbbreviation_IDX on LanguageFamilyAbbreviation in LanguageFamily
- Language_LanguageAbbreviation_IDX on LanguageAbbreviation in Language
- Server_CompleteServerName_IDX on CompleteServerName in Server
- ResourceVersion_ResourceID_CopyLocation_IDX on (ResourceID, CopyLocation) in ResourceVersion
- Subject_ResourceVersionID_Line_Col_IDX on (ResourceVersionID, Line, Col) in Subject
- UWEMTest_UWEMTestName on UWEMTestName in UWEMTest
- BarrierComputation_BarrierComputationName on BarrierComputationName in BarrierComputation

The indexes chosen reflect which SELECT [...] WHERE [...] expressions are used in the ETL tool and from stored procedures that – for efficiency – access the *datastaging* schema. Indexes have thus been created for (groups of) attributes that are used in the WHERE parts.

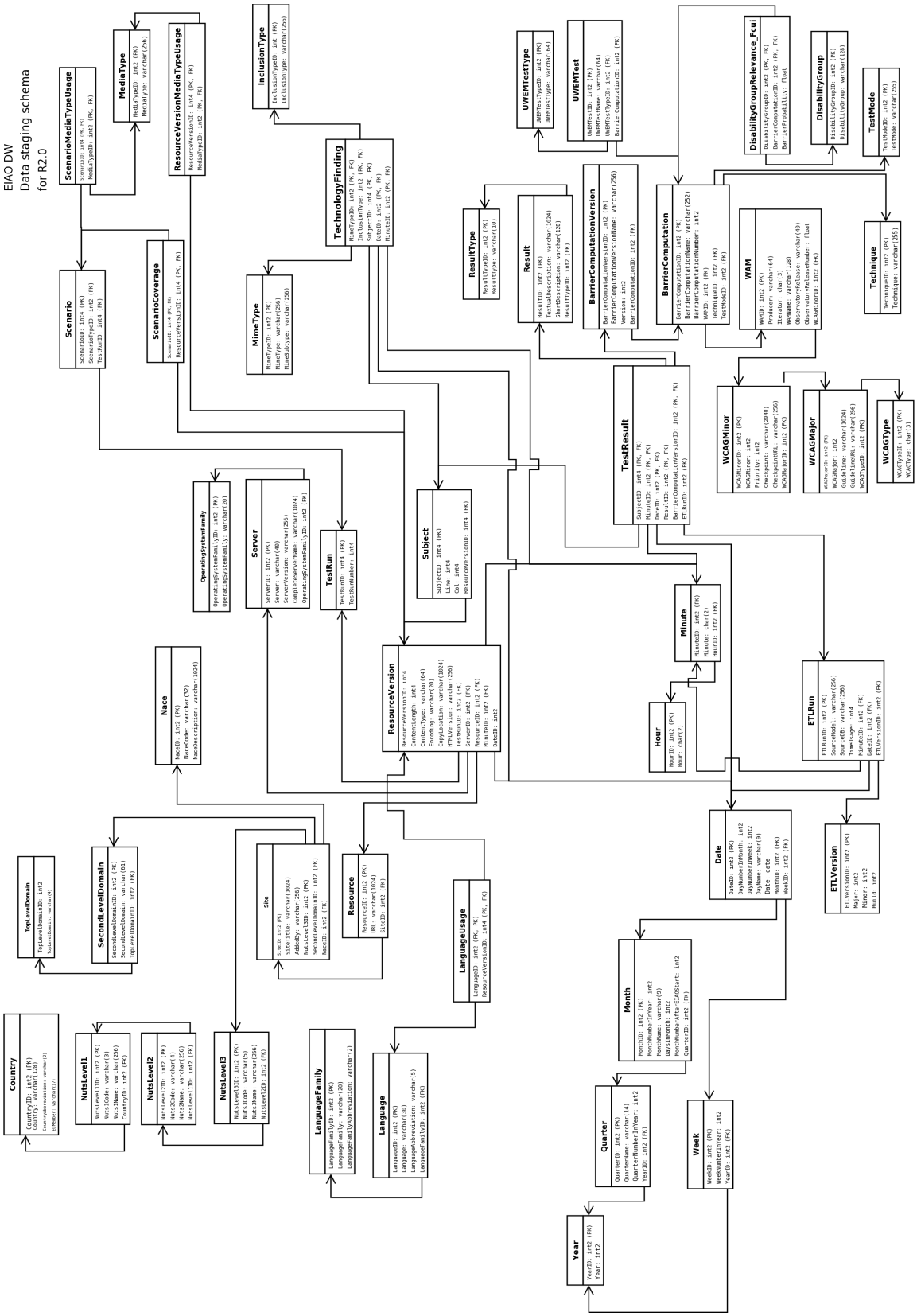

Figure 5.1: The data staging schema

Finally, a number of sequences have been added. These are used to ensure that ID attributes are assigned unique values. The sequences are used by the ETL to obtain the next free ID to use for a given table. The sequence for an ID attribute *X*ID is named *X*ID_seq. But for each of the Subject.SubjectID and Scenario.ScenarioID attributes, there are two sequences. For these two attributes there are also special sequences used by the ETL tool, but which are incremented with 10,000 each time nextval is called. Then the ETL tool can assign the values within this bound. This is due to the heavy use of the Subject and Scenario table in the ETL process.

The following sequences exist:
- NaceID_seq
- TopLevelDomainID_seq
- SecondLevelDomainID_seq
- SiteID_seq
- ResourceID_seq
- LanguageFamilyID_seq
- LanguageID_seq
- ServerID_seq
- ResourceVersionID_seq
- SubjectID_seq (maxvalue=1073741823)
- SubjectID_seq_for_ETL_tool (increment by 10000, minvalue=1073741824)
- MediaTypeID_seq
- ScenarioID_seq (maxvalue=1073741823)
- ScenarioID_seq_for_ETL_tool (increment by 10000, minvalue=1073741824)
- BarrierComputationVersionID_seq
- ETLRunID_seq
- ETLVersionID_seq
- MimeTypeID_seq

5.1 The ETL process

In the following, it is roughly described how the ETL process takes place.

5.1.1 Schema creation and prefilling of tables

Before the ETL starts, the used schemes should be created. This is done by using the createeiaodwr20 script (available from http://svn.eiao.net/robacc/Datawarehouse/). When this script is used, certain tables in the datastaging schema will be filled with data. The data to preload is located in tab-separated files in the http://svn.eiao.net/robacc/Datawarehouse/prefilldata/ directory. The tables being prefilled are
- BarrierComputation
- Country
-
- Hour
- InclusionType
- Minute
- Nace
- NutsLevel1
- NutsLevel2
- NutsLevel3

- OperatingSystemFamily
- Result
- ResultType
- ScenarioType
- Technique
- TestMode
- UWEMTest
- UWEMTestType
- WAM
- WCAGMinor
- WCAGMajor
- WCAGType

For each of these tables, there is a file (with the same prefix and ".csv" as suffix in the http://svn.eiao.net/robacc/Datawarehouse/prefilldata/) holding the data to preload. Some of these files are automatically generated. This is done by the Python scripts in http://svn.eiao.net/robacc/Datawarehouse/prefilldata/generators/. The auto-generated files are

- BarrierComputation.csv
- Country.csv
-
- Hour.csv
- Minute.csv
- NutsLevel1.csv
- NutsLevel2.csv
- NutsLevel3.csv
- Result.csv
- UWEMTest.csv
- WAM.csv

5.1.2 Loading the datastaging schema

The data in the RDF database is loaded into the data warehouse by the dw20load Python script in http://svn.eiao.net/robacc/Datawarehouse/. This program does the main part of the loading. The program extracts triples from the RDF database and inserts rows into the data warehouse. This is done by using a specialised RDFreader2 module in http://svn.eiao.net/robacc/RDFGenerator/RDFreader2.py. This module reads the RDF part of the EIAO RDF databases using regular expressions. Additionally, the dw20load program uses the psycopg package for connecting to the PostgreSQL database.

The program works by obtaining lists of triples from the RDF database and then iterating through these lists. The program first obtains a list of test runs and based on the data in this list the rest of the needed data is extracted. A significant part of the ETL code is in the method __handleTestRuns. The list of test runs is iterated, and for each test run it is considered if this test run should be included (this is done by comparing the test run number to the arguments given by the user). If a test run should be included, the method __addTestRuns is called. This method will add information about this test run to the DW and return an ID for the test run.

For each test run that is included, a list of site surveys performed in the test run in question is obtained. If a number of sites to handle are given to the dw20load program, it will only

consider the given sites. Otherwise, all sites in the test run will be considered. The program fetches an ID for a site to consider. This is done by calling the method __addSiteData. This method will look in the DW to see if the site is already represented, and if it is, return its ID. If not, __addSiteData will add the needed informations. To add information about the site to the DW, the __addSiteData method has to find out if the second level domain and top level domain are represented. This is done by calling __addDomainData (that itself looks for a representation of the domain parts and adds them if needed). This is the general pattern used by the __addXXXData in the loader. Note that the __addXXXData not necessarily have to look for data in the DW as they to a large extend cache results in main memory to increase the speed as explained later. The __addXXXData methods are:

- __addDataAndTimeData
- __addTestRunData
- __addDomainData
- __addSiteData
- __addPageData
- __addServerData
- __addLanguageData
- __addPageVersionData
- __addTechnologyFinding
- __addSubjectData
- __addMediaTypeFinding
- __addScenarioData
- __addScenarioMediaType
- __addBarrierComputationVersion
- __addETLRun

For each site seen by the __handleTestRunData method, it fetches scenarios for this site. And for each scenario, it fetches ranges that again can be used to fetch page surveys and assertions. So as hinted here, the __handleTestRunData method consists of many nested loops.

To increase the performance, main memory caches are used. Such a cache can, for example, hold the ResourceVersionID, Line, and Col, and SubjectID attributes for a number of subjects. It is then possible to detect if the needed data is present and find the needed ID without spending time on executing an SQL query. So instead of performing the query SELECT SubjectID FROM Subject WHERE ResourceVersionID = x AND Line = y AND Col = z on the database, a fast look-up in the cache (which is a hash mapping from (SubjectID, Line, Col) to SubjectID) can be done. Only if the cache does not hold the searched ID, it is necessary to execute the SQL query in the database. The caches, of course, only hold a fixed (but large) number of items.

Another mean to improve the performance is to write data to temporary files and use the COPY functionality of PostgreSQL later on to bulk load the data. This can be done for tables that are not being read from in the rest of the load process.

When the datastaging schema has been loaded, the primary key and foreign key constraints can be added to check that all expected constraints actually hold. The constraints are declared in the file http://svn.eiao.net/robacc/Datawarehouse/AddDataStagingConstraints_R20.sql.

During the ETL operation, C-WAM results for scenarios are also added to the matviews schema. These results are already available in the RDF data and by putting them into the summary tables in the matviews schema, the results can be reused by the stored procedures implementing the C-WAMs instead of doing the possibly expensive calculations from scratch. Further, different results for sites are put into the summary tables by the ETL.

5.1.3 Filling the eiaodw schema

When the datastaging schema has been loaded, the eiaodw schema can be loaded. The datastaging schema is partly a snowflake schema whereas the eiaodw schema is a star schema. So to load the data from the datastaging schema into the eiaodw schema requires some JOINs on the data in the datastaging schema. The SQL commands in http://svn.eiao.net/robacc/Datawarehouse/FillEIAODW_R20.sql perform INSERTs of the results of SELECT queries into the tables in eiaodw. Finally the constraints for the eiaodw schema can be added by using the file http://svn.eiao.net/robacc/Datawarehouse/AddEIAODWConstraints_R20.sql.

5.1.4 Filling the matviews schema

When the eiaodw schema has been loaded, the matviews schema can be loaded. This should be done *before* the stored procedures are used (i.e., this should be done *before* the GUI connects to the DW). The reason for this is that for efficiency reasons some of the stored procedures only look for data in the materialized views.

The SQL statements in the file http://svn.eiao.net/robacc/Dataewarehouse/FillMatViews.sql are used to fill the materialized views. These statements perform SELECTs (including JOINs) on the data in eiaodw and INSERTs to the matviews schema. Note that the FillMatViews.sql file currently contains hardcoded test run IDs. These IDs should be updated to the correct ID before the file is used.

6 C-WAM implementation in stored procedures

In this chapter, the implementation of the C-WAMs are described. The C-WAMs are implemented as stored procedures in the data warehouse (in the *eiaodw* schema). This is done to gain a good performance when doing potentially large aggregations.

6.1 The implemented procedures

The stored procedures used by the GUI are described in the following. It should be noted that many of the stored procedures are given a DisabilityGroupID argument. In the current implementation this argument is ignored as the C-WAM specification D3.3.2 does not use the concept of disability groups anymore. However, they are still given to the stored procedures to allow custom-implementations of other C-WAM calculations where an accessibility issue can have different "severity scores" for a blind and deaf user. When the current C-WAMs are used, the value 0 should be used for the DisabilityGroupID to indicate that the result should be for the "All" group. It should also be noted that some auxiliary procedures are defined, but not described below as they are not supposed to be called by the end-user/the GUI. For the full details of the implementation, the reader is referred to http://svn.eiao.net/robacc/Datawarehouse/StoredProcedures.sql.

Name: CWAM_Scenario
Parameters: *ArgScenarioID*, *ArgDisabilityGroupID, TestRunID*
Returns: The barrier probability (as defined in D3.3.2) for the page with the ID *ArgScenarioID* (including its used CSS files) in the test run with ID *TestRunID*.

Name: DPageTestRatio
Parameters: *ArgScenarioID, ArgBarrierComputationID, ArgTestRunID*
Returns: The fraction of FAIL outcomes of the barrier computation with ID *ArgBarrierComputationID* on the page with ID *ArgScenarioID* (and including its used CSS files) in the test run with ID *ArgTestRun*.

Name: DSite{Mean|Stddev|ErrorMargin|Min|Max}
Parameters: *ArgSiteID, ArgDisabilityGroupID, ArgTestRunID*
Returns: The {Mean|Stddev|ErrorMargin|Min|Max} for the accessibility scores within the site with ID *ArgSiteID* in the test run with ID *ArgTestRunID*.

Name: DPageList
Parameters: *ArgSiteID, ArgTestRunID*
Returns: An array of scenario IDs for the pages (and their CSSs) from the site with ID *ArgSiteID* in the test run with ID *ArgTestRunID*.

Name: DSiteTestRatio
Parameters: *ArgSiteID, ArgBarrierComputationID, ArgTestRunID*

Name: DSiteTestRatio
Returns: The fraction of FAIL outcomes of the barrier computation with ID *ArgBarrierComputationID* on the site with ID *ArgSiteID* in the test run with ID *ArgTestRun*.

Name: DPageContent
Parameters: *ArgScenarioID, ArgMimeTypeID, ArgInclusionTypeID, ArgTestRunID*
Returns: The number of usages of the MIME type with ID *ArgMimeTypeID* included in the way with ID *ArgInclusionTypeID* in the page with ID *ArgScenarioID* (including used CSSs) in the test run with ID *ArgTestRunID*.

Name: DSiteContent
Parameters: *ArgSiteID, ArgMimeTypeID, ArgInclusionTypeID, ArgTestRunID*
Returns: The number of usages of the MIME type with ID *ArgMimeTypeID* included in the way with ID *ArgInclusionTypeID* in the site with ID *ArgSiteID* in the test run with ID *ArgTestRunID*.

Name: DGroup{Mean|Stddev|ErrorMargin|Min|Max}
Parameters: *ArgGroup* (an array of SiteIDs), *DisabilityGroupID, TestRunID*
Returns: The {Mean|Stddev|ErrorMargin|Min|Max} for the accessibility scores within the sites with IDs in *ArgGroup* in the test run with ID *ArgTestRunID*.

Name: DGroupTestRatio
Parameters: *ArgGroup* (an array of SiteIDs), *ArgBarrierComputationID, ArgTestRunID*
Returns: The fraction of FAIL outcomes of the barrier computation with ID *ArgBarrierComputationID* on sites with IDs in *ArgGroup* in the test run with ID *ArgTestRun*.

Name: DGroupContent
Parameters: *ArgGroup* (an array of SiteIDs), *ArgMimeTypeID, ArgInclusionTypeID, ArgTestRunID*.
Returns: The number of usages of the MIME type with ID *ArgMimeTypeID* included in the way with ID *ArgInclusionTypeID* in the sites with IDs in *ArgGroup* in the test run with ID *ArgTestRunID*.

Name: NutsGroup
Parameters: *ArgNutsCode, ArgTestRunID*
Returns: An array of SiteIDs of sites with the NUTS code *ArgNutsCode* and tested in the testrun with ID *ArgTestRunID*.

Name: EUMembershipGroup
Parameters: *ArgEUMember, ArgTestRunID*

Name: EUMembershipGroup
Returns: An array of SiteIDs of sites from countries with EU membership as in *ArgEUMember* ('EU member', 'Applicant country' Outside EU', 'All') in the test run with ID *ArgTestRunID*.

Name: NaceGroup
Parameters: *ArgNaceCode, ArgTestRunID*
Returns: An array of SiteIDs for sites belonging to the NACE group *ArgNaceCode* in the test run with ID *ArgTestRunID*.

Name: AllGroup
Parameters: *ArgTestRunID*
Returns: An array of all sites evaluated in the test run with ID *ArgTestRunID*.

Name: NaceNutsGroup
Parameters: *ArgNaceCode, ArgNutsCode, ArgTestRunID*
Returns: The intersection of NaceGroup(*ArgNaceCode, TestRunID*) and NutsGroup(*ArgNutsCode, ArgTestRunID*).

Name: getMimeTypeID
Parameters: *ArgMimeType*
Returns: The MimeTypeID for the the MIME type *ArgMimeType*.

Name: getInclusionType
Parameters: *ArgInclusionTypeID*
Returns: The InclusionTypeID for the inclusion type *ArgInclusionType* ('Internally linked', 'Externally linked', 'Embedded', 'Unknown').

Name: getBarrierComputationID
Parameters: *ArgBarrierComputationName*
Returns: The BarrierComputationID for the barrier computation with the name *ArgBarrierComputationName*.

Name: getSiteID
Parameters: *ArgSite*
Returns: The ID for the site *ArgSite*.

7 A Note on The EIAO DW Specification

7.1 The Challenge of EIAO DW Specification

A DW is basically a system that structures:

a) A certain set of data
b) In a certain way, to answer
c) A certain set of queries
d) In an easy, efficient, and scalable way.

That means that a good, precise DW specification should ideally contain a precise specification of all items a)-d). Since item b) is heavily dependent on a) and c), b) can only be specified precisely if a) and c) can be specified precisely. For item d), some things can be specified without knowing the full details of the other items, e.g., the database size to be supported in GB can be specified. Other things, e.g., response time requirements, cannot be specified meaningfully if c) (the queries) are not known precisely.

In "ordinary" DW development projects, the source (operational) systems are completed and running before the DW project is even started. Also, before the DW project is started, there is usually a good understanding of what kind of answers (queries) one would like to get from the DW. That means that all items a)-d) can be specified precisely at the start of the DW project.

For the EIAO DW, things are quite different, as the "operational system" (crawler, WAMs, repository) is being developed more or less at the same time as the DW.
The challenge with the EIAO DW specs is here that WP6 is at the end of a very long pipeline: UWEM->WP3->WP5->WP6.
For example, a very high-level description of a) and c) can be derived from UWEM, but it is only in D3.3.2 that a precise definition of the source data is given, and the actual format of the source data must also be available and realistic example source data is also required. That means that very important input to a precise DW specification was only available during the development. This requires the development of the DW to be done in an incremental way.

We will now go on to briefly discuss various qualities of the EIAO DW R2.2 software.

7.2 Modularity

First we will consider the modularity of the EIAO DW R2.2. As seen above, the different parts of the DW DB are kept separate, e.g., the DW schema for user queries are separate from the "internal" parts used only by the ETL. The ETL code is well structured and modular, as can be seen in the documented source code. The user DW schema and the queries on it are isolated by the use of stored procedures for the CWAM queries. The CWAM stored procedure code is very modular and general, and can easily be extended with new criteria and groupings with very little effort.

7.3 Extensibility

Next we consider the extensibility of the DW, meaning the ability to introduce significant new types of data and queries without "disturbing" (having to change) the existing data and queries.

The dimensional modeling approach used in the EIAO DW is known to be very robust, as it allows new fact tables, measures, dimensions, dimension levels, and dimension to be introduced "gracefully" without problems. For example, additions of new WAMs require no (or very little change) to the DW schema and the ETL, as both of these model WAMs in a generic way. This is witnessed by the smooth integration of Imergo WAM data in the R1.0 DW which modeled WAMs in a very similar way. It is also easy to introduce completely new fact and dimension tables as was done in R2.0 where, e.g., the MediaType dimension and TechnologyFinding fact table were added.

7.4 Scalability

We now consider the scalability of the DW, in terms of data volumes and growth over time. The chosen DBMS, PostgreSQL, is widely used around the world, and has proven to work well for very large databases. We have developed a strategy for partitioning the DW per testrun (per month). Each partition can then be loaded, indexed, and queried (as C-WAM queries generally only concern the most recent testrun, and perhaps a past reference testrun) separately and transparently. As the size of a partition is not growing over time (as is the total DW size), partitioning can ensure that the DW can grow over time without problems and such that the time usage for the C-WAM calculations does not increase.

If certain queries have performance problems, materialized views can be added transparently because of the stored procedures used for DW queries. Further, the stored procedures exploit caching of already calculated results.

7.5 Maintainability

Finally, we consider the maintainability of the DW software, meaning the ability to carry out small, incremental changes to the software.

The code is easy to follow and documented. The same is true for the DB schema. The ETL code closely mimics the repository RDF schema, and is thus easy to understand for other EIAO developers. All database tables can be maintained incrementally without taking down the system.